XLEngine : Une architecture cross-layer modèle pour le support de la QoS dans les réseaux sans fil IEEE 802.11

Wafa Berrayana, Habib Youssef, Guy Pujolle, Stéphane Lohier

XLEngine : Une architecture cross-layer modèle pour le support de la QoS dans les réseaux sans fil IEEE 802.11

Wafa Berrayana , Habib Youssef , Guy Pujolle , Stéphane Lohier

***Résumé*— Initialement les réseaux locaux sans fil IEEE 802.11 ont été conçus pour mettre en place des transmissions dans des endroits difficiles à câbler et assurer la communication des données pour des applications mobiles. Ces réseaux doivent actuellement supporter des applications plus complexes ayant des besoins contraignants en terme de délai de bout en bout garanti, de bande passante disponible, etc. Pour apporter cette Qualité de Service (QoS), une des solutions émergeantes consiste à développer des approches *cross-layer* permettant aux différentes couches de coopérer ensemble afin de mieux supporter la QoS. La littérature nous présente une multitude d'architectures selon ce principe, mais aucune de celles-ci n'est normalisée. De plus, suivant les besoins à satisfaire, de nouvelles architectures sont régulièrement proposées, mettant en œuvre de nouveaux protocoles ou des adaptations de ces derniers. Par conséquence, nous risquons d'avoir, à long terme, des systèmes sans fil incompatibles développés selon différentes architectures *cross-layer*. Dans ce papier, nous proposons des éléments de base d'une architecture *cross-layer* modèle qui soit paramétrable, extensible, flexible et générique pour contribuer au support de la QoS dans les réseaux sans fil IEEE 802.11.**

***Mots clés*— réseaux sans fil IEE 802.11, ad-hoc, QoS, conception *cross-layer*, modèle en couches, etc.**

I. Introduction

Les propriétés dont disposent les réseaux sans fil par rapport aux réseaux filaires imposent un nouveau défit lors de la conception de protocoles sans fil. Ces derniers doivent être dynamiquement adaptatifs aux changements continus de l'état du canal radio [1] et doivent au même temps chercher à satisfaire les exigences en QoS des applications supportées. Dans [2], les auteurs considèrent que pour atteindre ces objectifs, les protocoles conçus doivent satisfaire les propriétés clés suivantes : 1) correspondance des critères de QoS entre les différentes couches et optimisation de la QoS à travers ces couches; 2) adaptabilité suffisante aux variations de l'état du canal; 3) robustesse aux pertes de données; 4) support du facteur d'échelle pour la bande passante et les équipements.

Dans [3], [4], [5] les auteurs considèrent que les exigences de QoS ne peuvent être satisfaites si les solutions proposées se focalisent sur des couches particulières de la pile protocoloaire sans se préoccuper des autres aspects liés aux autres couches, ni à la nature même des réseaux sans fil en termes de variations continues dans le temps et dans l'espace. Ces différentes études illustrent et justifient la tendance actuelle qui consiste à développer des approches inter-couches permettant aux différentes couches de coopérer ensemble afin de supporter la QoS tout en considérant les spécificités du canal radio. Cette notion *'cross-layer'* (nous gardons cette terminologie anglophone clé qui signifie inter-couches ou multi-couches) implique 'une cassure' de la notion de 'couches isolées' et un échange d'informations entre toutes les couches, éventuellement non adjacentes, selon une méthodologie non encore normalisée.

Ainsi une question fondamentale s'impose : quelle est l'architecture la plus adéquate pour ces échanges et quels sont les éléments de bases de cette architecture ?

Répondre à cette question est notre objectif tout au long de ce papier. Nous proposons des éléments de base d'une architecture cross-layer modèle qui soit paramétrable, extensible, flexible et générique pour le support de la QoS dans les réseaux sans fil IEEE 802.11.

Le reste de ce papier est organisé comme suit. Dans la section II, nous présentons une étude des solutions et architectures *cross-layer*. Dans la section III, nous discutons de l'importance d'une architecture *cross-layer* modèle de référence et nous présentons les principales caractéristiques de cette architecture. La section IV présente notre architecture *cross-layer* développé selon ce principe. La section VI conclue le papier et présente quelques perspectives.

II. Etude cross-layer et architectures existantes

A. Etude cross-layer

L'étude des solutions d'optimisation et d'adaptation *cross-layer* [3], [4], [5], [6] est une voie de recherche émergeante offrant un grand potentiel et une grande diversité afin d'améliorer les performances des réseaux sans fil. Parmi ces solutions, nous citons l'exploitation de la mesure des durées de veille du canal sans fil (couche liaison) pour améliorer le choix des routes (couche réseau) dans un contexte ad-hoc [7], ou encore l'adaptation des temps d'accès au support (couche liaison) aux contraintes de délai et de bande passante (couche transport) pour des applications temps réel [7]. Dans [7], l'auteur adopte une approche *cross-layer*, d'une part afin d'étudier et d'améliorer les performances des flux élastiques

W. Berrayana est doctorante associée aux laboratoires Prince, Hamman Sousse, Tunisie et lip6, Paris, France. E-mail: wafa.berrayana@lip6.fr.

H.Youssef est professeur à l'université de Sousse, Tunisie et associé au laboratoire Prince, Hammam Sousse, Tunisie. E-mail: habib.youssef@fsm.rnu.tn .

G.Pujolle est professeur à l'université de Pierre et Marie Curie, Paris et directeur de l'équipe Phare du laboratoire lip6, Paris, France. E-mail: guy.pujolle@lip6.fr.

S. Lohier est enseignant à l'université de Marne la vallée, Paris et docteur du lip6, Paris, France. E-mail: stephane.lohier@lip6.fr.

TCP présents dans la plupart des applications de l'Internet sur le dernier lien sans fil. D'autre part, pour étendre et incorporer un mécanisme de notification de perte de QoS au niveau du protocole de routage AODV. Ainsi, le choix des routes selon AODV se fait suivant les métriques de délai maximal de bout en bout et de bande passante minimale requise. Ces métriques sont évaluées au niveau de la couche MAC et sont exploitées par AODV pour servir de critère au routage QoS.

B. Architectures cross-layer existantes

Pour la mise en place des solutions *cross-layer*, trois catégories d'architectures sont proposées [8], [9] se classifiant comme le montre la figure 1.

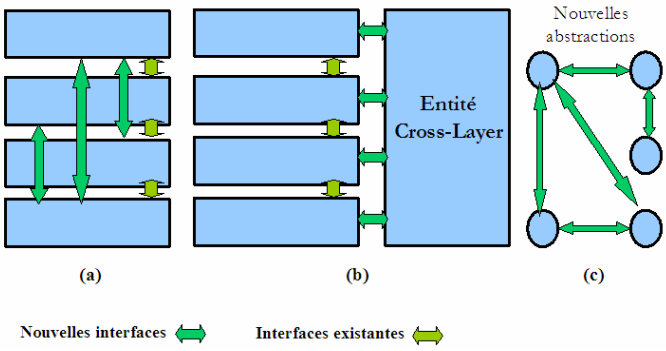

Fig. 1. Classification des architectures *cross-layer*

Pour la première catégorie d'architectures (cf. Fig.1.a), les couches, même non adjacentes, peuvent coopérer ensemble et communiquer directement pour assurer des optimisations *cross-layer* et fournir la QoS nécessaire. Dans ce but, de nouvelles routines sont à intégrer aux couches pour permettre la réception et le traitement des données *cross-layer*. Le sens des échanges de ces données ainsi que les couches impliquées dans cet échange [8] ne sont pas normalisés, ils dépendent de l'objectif QoS à atteindre. Nous pouvons affirmer qu'il y'a autant de variantes d'architectures selon ce principe que de protocoles et de buts à satisfaire.

Toutefois, cette approche entraîne un certain nombre de contraintes. D'un coté, l'extra code, dit code *cross-layer*, ajouté ralentit l'exécution du code original du protocole de la couche, réduisant ainsi son débit, surtout lorsqu'elle interagit avec plusieurs couches simultanément. D'un autre coté, la mise à jour du code *cross-layer* est difficile et délicate puisqu'il se fusionne avec le code original de la couche. Un autre point à souligner est que plusieurs optimisations *cross-layer* au niveau d'une même couche peuvent entraîner des conflits au détriment deà la justesse des résultats [10], [11]. Pour toutes ces limitations, les auteurs dans [10] considèrent que l'amélioration des performances selon cette catégorie d'architectures *cross-layer* est une approche forcement à court terme.

Pour la deuxième catégorie d'architectures *cross-layer* (cf. Fig.1.b), l'idée de base est de conserver le fonctionnement normal de la pile protocolaire tout en permettant des interactions *cross-layer* via une entité intermédiaire dont la dénomination et les fonctionnalités exactes varient selon l'architecture [12], [13], [14]. Les architectures selon ce principe permettent de :

- garder une compatibilité avec l'architecture classique en couches;
- maintenir les avantages de l'architecture modulaire en couches isolées offrant ainsi un environnement robuste de mise à jour, permettant l'addition et/ou la suppression des protocoles des différentes couches, sans avoir à modifier les autres couches;
- garantir une évolution continue de l'entité *cross-layer* en lui incorporant d'autres fonctionnalités sans gêner les protocoles relatifs aux couches;

Et pour finir, la troisième catégorie d'architectures *cross-layer* (cf. Fig.1.c) est particulièrement distincte des deux autres car elle présente des abstractions complètement nouvelles [16]. Elle adopte le concept du *'non-layered protocol architecture'*. La motivation pour cette architecture est la forte flexibilité obtenue avec un minimum de problèmes d'interactions et de communications entre ses différents composants.

Jusqu'à nos jours, plusieurs architectures *cross-layer* ont été proposées et sont déjà implémentées. Chacune se classifie forcement dans l'une des catégories exposées par la Figure 1. Comme exemples de ces architectures nous citons : CrossTalk [12], ECLAIR [13], MobileMan [14], CATS [17], WIDENS [18], GRACE [16]. Dans ce qui suit nous présentons quelques unes de ces architectures.

1) CrossTalk: L'objectif primordial de l'architecture CrossTalk [12] consiste à adapter le comportement d'un nœud à l'état global du réseau. Pour le faire, cette architecture maintient deux entités intermédiaires séparées, la première *local_View* qui détient une connaissance locale du nœud ainsi que les opérations *cross-layer* et la seconde *global_View* pour avoir une vue globale sur les autres nœuds du réseau. Toutefois, CrossTalk présente deux limitations principales. La première est les procédés complexes de collectes des informations locales des nœuds du réseau pour déduire une vue globale sur le réseau. La seconde est la complexité de l'architecture elle-même. Cependant, d'après une comparaisons faite dans [12], cette architecture présente de meilleurs performances par rapport aux architectures précitées.

2) ECLAIR: Les composants principaux de ECLAIR [13] sont le sous système d'optimisation (OSS) qui constitue le moteur *cross-layer* et les 'Tuning layers' (TL). Le OSS contient les optimisateurs de protocoles (PO) qui maintiennent les algorithmes *cross-layer*. Les TLs fournissent les APIs des POs pour l'interaction avec les couches et la manipulation des structures de données correspondantes. Chaque protocole du modèle en couches dispose de son propre TL. L'avantage de ECLAIR c'est sa grande capacité de support de plusieurs types d'applications. Cependant elle est caractérisée par sa grande complexité étant donné que les interactions et les optimisations *cross-layer* passent par plusieurs entités. De plus, ECLAIR influence fortement le comportement des protocoles.

*3) **MobileMAN**:* [14] décrit l'architecture de MobileMAN. Le composant *cross-layer* de MobileMAN est dénoté *Network Status*, il maintient les informations *cross-layer*. L'accès à cette entité est bien défini pour faciliter l'accès aux données par les protocoles.

Pour bien exploiter les avantages de l'architecture MobileMAN et exploiter les informations *cross-layer*, les protocoles

du modèle en couches doivent être re-conçus. Au niveau de *Network Status*, les données sont exprimées sous une forme abstraite pour être compréhensibles et exploitables par tous les protocoles. Cette abstraction ajoute de la complexité à l'entité *Network Status*. De l'autre coté, MobileMAN dispose d'un talent pour résoudre des problèmes propres au réseau ad-hoc (e.g. problème la gestion de l'énergie). Cependant MobileMAN présente des limitations. Elle limite le nombre de protocoles tournant au sein du framework. De plus, MobileMAN ne prévoit pas une vue globale sur le réseau et principalement ne fournit pas des décisions *cross-layer* optimales.

4) ***CATS: Cross-layer Approach to Self-healing*** *:* L'architecture CATS [17] est ajustée aux réseaux de senseur. Elle est caractérisée par la présence d'un composant nommée *Management Plane* maintenant les protocoles et visibles par toutes les couches. Le *Management Plane* est un composant actif influençant le comportement des protocoles et même les paquets traversant la pile protocolaire (e.g. le Management Plane peut changer l'adresse de destination d'un paquet ou encore influencer le protocole de routage pour stopper la réponse aux requêtes de routes). En conséquence, l'inconvénient de CATS est que le framework lui-même représente un composant actif. Ce qui signifie que l'ajout ou la suppression d'un protocole ou d'une application doit être suivi par une mise à jour complète du framework. Cette architecture introduit de la complexité à la conception, à la fois, des protocoles et de l'architecture.

5) WIDENS: WIDENS [18] est une architecture développée pour les réseaux ad-hoc pour être déployée en cas d'urgence (e.g. catastrophes naturels). L'idée fondamentale de WIDENS consiste à sélectionner certains protocoles pour tourner dans l'architecture tout en gardant le principe de généricité. WIDENS s'intéresse principalement aux protocoles de routages sous des contraintes de QoS. Les interactions *cross-layer* s'effectuent via des interfaces bien définies entre les couches adjacentes. Ces extensions *cross-layer* fournissent des informations d'état et un mappage de paramètre pour augmenter le pouvoir de re-configuration et l'adaptabilité des protocoles. Les informations *cross-layer* sont exploitées lorsque les optimisations isolées au niveau d'une couche n'empêchent pas la dégradation des performances. Les boucles d'adaptations potentielles sont évitées seulement en tolérant les interactions entre les couches adjacentes. Les informations des couches non adjacentes sont seulement exploitées via des fonctions de mappage de la couche en bas ou en haut. Ceci seulement est accompli si les adaptations *cross-layer* au niveau de la couche adjacente sont inefficaces et les autres couches doivent s'adapter en addition pour satisfaire les performances.

Le domaine d'intérêt de l'architecture WIDENS est très retreint puisqu'elle s'intéresse à un domaine d'application spécifique. Le *cross-layering* est très peu exploité, par rapport à d'autres architectures *cross-layer* tel que MobileMAN, puisque uniquement les couches adjacentes exécutent les opérations *cross-layer* et maintiennent les interactions *cross-layer*. Ceci dégrade les performances de WIDENS puisque les couches voisines peuvent ne pas êtres le choix optimal pour une bonne adaptation. De plus, l'optimisation n'est pas un objectif principal pour cette architecture.

C. Etude comparative entre les architectures cross-layer existantes

Nous présentons dans cette section une étude comparative entre les architectures *cross-layer* précitées. Comme critères de comparaisons nous citons [12] :

- capacité de support de nouvelles applications : elle dénote la "générécité" de l'architecture (très importante ++ → très faible - -) ;
- complexité de fonctionnement : elle reflète le niveau de complexité ajouté par l'architecture pour la mise en place des optimisations et des échanges de données *cross-layer* (très importante - - → très faible ++) ;
- complexité de gestion des données : ce critère quantifie l'effort nécessaire pour la représentation, la sauvegarde, le partage et l'accès aux données *cross-layer* (très importante - - → très faible ++) ;
- complexité d'adaptation des protocoles : elle dénote la complexité induite pour l'adaptation des protocoles du modèle en couches à l'architecture cross-layer (très importante - - → très faible ++) ;
- capacité d'optimisation à l'échelle du réseau (très importante ++ → très faible - -) ;
- les problèmes relatifs aux réseaux ad-hoc et les solutions proposées par les architectures pour en faire face ;
- amélioration du modèle en couches ;

La table suivante [12] présente une comparaison entre les architectures CATS, WIDENS, ECLAIR, MobileMan et CrossTalk selon les critères précités. Ces critères sont pondérés dans la table suivante selon cette légende :

+ + : très importante, très faible
+ : bien
0 : moyenne
- : importante, faible
- - : très importante, très faible

Critère / Architecture	Généricité de l'architecture	complexité de fonctionnement	complexité de gestion des données	capacité d'optimisation à l'échelle du	Complexité d'adaptation des protocoles
CATS	-	-	-	+	0
WIDENS	0	0	-	0	-
ECLAIR	++	--	-	+	--
MobileMan	++	+	0	+	+
CrossTalk	++	0	0	++	+

Fig. 2. Comparaison entre les architectures *cross-layer* proposées

III. Importance d'une architecture *cross-layer* modèle

Il faut noter que la conception cross-layer n'est pas un simple remplacement de l'architecture en couches, ni une simple combinaison de leurs fonctionnalités. Le processus de partage de données et de paramètres doit être bien structuré et l'architecture *cross-layer* doit être conçue efficacement car

toute conception inadaptée pourrait potentiellement aggraver le problème de performance (support de la QoS) qu'elle vise à résoudre. Un exemple illustré par des simulations confirme cette idée [7]. Il prouve que les protocoles MAC tendant à accroître le taux de transfert de trame (rate-adaptive MAC protocols) lorsque le canal est performant (rapport SNR élevé) peuvent entraîner une baisse du débit sur les routes multi sauts. Il est donc impératif, en choisissant une architecture *cross-layer* et en développant une solution *cross-layer* dans le cadre de cette architecture, d'essayer de pronostiquer toutes les interactions non désirées en analysant systématiquement leurs impacts sur toutes les couches.

A. Challenges de la conception cross-layer

Nous distinguons de nombreux challenges [10], [11], [15] associés à la conception et à l'implémentation d'une architecture *cross-layer*, parmi lesquels nous citons :

- Dans certains cas (les architectures de la première catégorie, cf. Fig.1.a), l'implémentation d'une architecture *cross-layer*, peut entraîner des dépendances qui ne sont pas essentielles pour le bon fonctionnement du système entier. En effet, une interaction particulière considérée peut avoir des conséquences désastreuses sur l'exécution de la totalité du système. Les nouvelles interactions qui sont donc liées à la perte du privilège de conception sur une couche particulière doivent donc être étudiées en utilisant des graphes de dépendance.
- Avec une conception *cross-layer*, le nombre de nouvelles interactions peut augmenter, ce qui peut engendrer une re-conception complète du système entier et empêcher par la suite d'autres innovations. Dans [10], Kawadia et al développent ce problème et l'illustrent au travers d'exemples.
- Les modifications aux différentes couches peuvent avoir comme objectif l'optimisation d'une métrique qui leur est commune (e.g. délai) dans des directions opposées, ce qui peut avoir des conséquences négatives car la modification d'une métrique au niveau d'une couche peut avoir un impact implicite sur les autres.
- L'implémentation des différentes interactions *cross-layer* peut entraîner une déstructuration du code original des couches concernées. Le maintient, la mise à jour et la compréhension du code devient au fil du temps une tâche ardue à la fois pour le concepteur et l'utilisateur.
- Un autre défi très important à souligner porte sur la façon dont sont stockées les données *cross-layer*. Il faut étudier judicieusement les types de ces données et leurs influences, à la fois sur les couches et sur les algorithmes *cross-layer* à exécuter. Les stratégies de stockages des données doivent tenir compte des caractéristiques des réseaux sans fils en termes de limitations de ressources et de mobilité [11]. Le but est de permettre un accès rapide à ces données pour contribuer à la rapidité de leurs traitements.
- Un dernier défi à souligner concerne l'accès aux différentes couches. Il faut que cet accès soit rapide, en temps réel avec un minimum de surcharges induites.

B. Spécificités de l'architecture cross-layer modèle

L'architecture *cross-layer* proposée doit être d'une part :

- *générique* pour supporter une large gamme d'applications avec QoS;
- *extensible* et *flexible* pour suivre et supporter l'évolution continue des applications QoS et satisfaire à leurs nouvelles exigences;
- *paramétrable* en maintenant des composants génériques pour permettre le développement de nouveaux composants si ceux déjà existant demeurent incapables de répondre aux besoins de certaines applications. L'architecture doit permettre également une adaptation aux variations des paramètres d'optimisations des applications (e.g. énergie ou latence) et aux changements dans leur environnement (e.g. changement de la mobilité des nœuds, etc.).

D'autre part, l'architecture *cross-layer* doit assurer :

- *un prototypage rapide* qui consiste à incorporer de nouveaux algorithmes et/ou données *cross-layer* d'une façon transparente;
- *un minimum d'intrusion* permettant un interfaçage facile avec l'architecture modale en couches sans avoir à la modifier;
- *une portabilité* avec les autres systèmes sans fil.

Également, une architecture *cross-layer* doit offrir une vue générale sur le réseau puisque certaines opérations d'optimisations requièrent par nature de connaître certaines informations à partir d'autres nœuds du réseau pour prendre des décisions plus objectives.

Il est donc clair que le choix d'une architecture adaptée permettant, d'une part, le maintient des performances existantes et l'apport de la QoS requise par d'autres applications; et d'autre part, de faire faces aux défis précitées est une tâche ardue et complexe [10]. Dans le domaine *cross-layer* malgré la multitude et la diversité des architectures proposées (cf. section II), aucune architecture modèle n'est normalisée jusqu'à nos jours. Selon les besoins à satisfaire, de nouvelles architectures sont régulièrement proposées, mettant en œuvre de nouveaux protocoles ou des adaptations de ces derniers (e.g. AODV, OLSR, EDCF, etc.). Par conséquence, nous risquons d'avoir, à long terme, des systèmes sans fil incompatibles développés selon différentes architectures *cross-layer*.

Pour toutes ces raisons, nous soulignons l'importance de la proposition d'une architecture *cross-layer* modèle générique, extensible, flexible et paramétrable qui pourrait servir comme point de départ pour les conceptions *cross-layer*.

Dans ce papier, nous cherchons donc à élaborer cette architecture. Deux étapes sont envisagées. La première consiste à choisir parmi les catégories d'architectures précitées (cf. section II), celle qui semble la plus appropriée à la fois pour satisfaire les points évoqués et pour les spécificités des réseaux sans fil. La seconde consiste en la mise en place des composants de base de cette architecture.

IV. XLEngine : architecture modèle proposée

A. Catégorie de l'architecture cross-layer modèle

Compte tenu des objectifs du *cross-layering*, des problèmes engendrés par une architecture *cross-layer* inadaptée, et partant des architectures précitées, nous faisons le choix d'une architecture adoptant le principe de communication via une entité intermédiaire (cf. Fig.1.b). Les raisons qui motivent notre choix sont multiples. La première raison est qu'elle conserve les avantages de l'architecture en couches, particulièrement en terme de modularité, contribuant ainsi à la longévité et à la compatibilité avec le modèle OSI. De plus, la présence de l'entité *cross-layer* permet une évolution individuelle et continue à la fois pour les couches et pour l'entité elle même sans gêner le système global. Un autre avantage à souligner est que cette entité dispose d'un libre accès à toutes les couches, ce qui rend ses décisions plus objectives. Elle permet également une intégration facile et simple de nouveaux algorithmes et données *cross-layer* sans avoir à changer au reste de l'architecture. Finalement, la présence de cette entité évite toute duplication d'efforts par les différentes couches pour la recherche des informations sur les autres couches non adjacentes ou d'informations sur l'état du canal.

B. Principe de l'architecture XLEngine

Notre architecture XLEngine est développée selon quatre principes de bases :

- *Principe 1 :* supporter tous types d'applications et tous types de protocoles (protocoles de routage, protocoles d'accès au canal, etc) ;
- *Principe 2 :* masquer les détails de la procédure de recherche des données *cross-layer* par les protocoles du modèle en couches ;
- *Principe 3 :* apporter le minimum de modifications aux protocoles du modèle en couches ;
- *Principe 4 :* les optimisations *cross-layer* tiennent en considération l'état global du réseau. Notre motivation pour ce principe vient du fait qu'une décision basée sur une connaissance globale du réseau est toujours plus efficace que celle basée uniquement sur une connaissance locale [17].

Le quatrième principe, en outre des autres principes, constitue la distinction et l'apport principaux de XLEngine par rapport aux autres architectures *cross-layer*, excepté CrossTalk. En effet, la connaissance de l'état global du réseau permet au noeud d'évaluer son propre état à tout moment par rapport au réseau entier. Ceci permet au nœud de changer le comportement de ses protocoles et de ses stratégies d'optimisations pour répondre à des objectifs à l'échelle de son réseau comme le load balancing. Ceci lui permettra de prendre des décisions plus robustes pour, par exemple, ajuster sa charge à celle que le réseau peut supporter et éviter ainsi tout gaspillage possible de la bande passante. Dans ce contexte, nous présentons deux citations accentuant l'importance d'avoir une optimisation à l'échelle du réseau :

- *Citation 1 : "Due to the absence of global information, localized algorithms may not produce optimal or near-optimal solutions"* [17].
- *Citation 2 : "Locally available information such as local load, battery status or neighbour degree might not lead to optimal or even near-optimal results when used for decision processes such as routing. Global knowledge would enable a node to evaluate its relative state by comparing its local value against the global value of the respective metric"* [12].

Nous dénotons par *XLEngine (Cross-Layer Engine)* notre architecture *cross-layer* modèle proposée. Nous présentons dans ce qui suit les éléments de bases de cette architecture.

C. Présentation de XLEngine

L'entité de communication *cross-layer* de *XLEngine* (cf. Fig.3) est désignée *CLMC (Cross-Layer Management Component)*. C'est essentiellement via le *CLMC* que les couches et les applications communiquent. Elle comporte essentiellement deux parties : le *Cross-Layer Interface Component* (*CLIC*) et le *Cross-Layer Engine* (*CLE*).

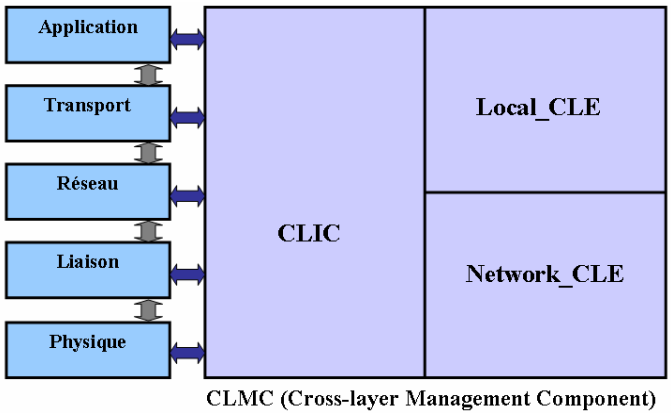

Fig. 3. XLEngine : Architecture *cross-layer* proposée

- **CLIC (Cross-Layer Interface Component) :** c'est le plan de contact entre les couches et les applications d'une part et CLMC d'autre part. CLIC se charge d'intercepter et de maintenir les données des protocoles des couches dans leurs formats bruts (e.g. état du canal décrit par le SNR fourni par la couche physique, le Retry limit fourni par la couche MAC, le nombre de sauts d'une route fourni par le protocole de routage, le TCP Window Size, les exigences en QoS fournies par la couche Application, etc.). Ces données nous les qualifions de *cross-layer* puisqu'elles pourront être utilisées par les autres couches pour des optimisations *cross-layer*. Cependant, si pour un certain protocole les données *cross-layer* dont il a besoin sont indisponibles temporairement par CLIC pour une certaine optimisation *cross-layer*, alors le protocole doit toujours accomplir ses fonctionnalités de base. L'indisponibilité des données *cross-layer* doit avoir comme unique résultat la dégradation des performances.
- **CLE (Cross-Layer Engine) :** c'est le moteur cross-layer de la CLMC. Il maintient deux parties : Local_CLE et Network_CLE.

– **Local_CLE :** CLIC maintient des données *cross-layer* de bases générées par les protocoles des couches. Local_CLE génère des données plus complexes à partir des données de CLIC. Ces métriques peuvent être requises par un ou plusieurs protocoles. Donc, au lieu que chacun génère individuellement ces métriques, c'est Local_CLE qui se charge de le faire. Comme exemple de ces métriques nous citons, le débit de transmission effectif d'une application [18]. Il est évalué sachant la bande passante au niveau des nœuds sauts d'une route et sachant le nombre de noeuds interférant avec les sauts de la route. Le taux de transmissions réussies (STR : Successful Transmission Rate) et le taux de tentatives pour trouver un canal disponible (CCR : Clear Channel Rate) sont d'autres métriques générées par Local_CLE. Le STR fournit une mesure de QoS sur les collisions, en excluant les congestions; tandis que le CCR fournit une mesure des congestion. Ces métriques peuvent ensuite servir au niveau de la couche physique pour ajuster la puissance de transmission radio ou encore au niveau de la couche réseau pour trouver des routes sans congestion ni collisions. Le fait de générer ces métriques au sein de Local_CLE et non pas au sein des couches physique et réseau a l'avantage de laisser les protocoles dédiés se focaliser sur leurs fonctionnalités de bases et de les alléger. Une autre spécificité des métriques générées par Local_CLE c'est qu'elles peuvent être déduites après une période d'observation des métriques de CLIC. C'est le cas, par exemple, des métriques STR et CCR. Les métriques évaluées par Local_CLE est dite métrique locale.
– **Network_CLE:** comme c'est déjà mentionné au niveau de la section précédente, une spécificité majeure de XLEngine c'est qu'elle maintient les données *cross-layer* des autres nœuds du réseau dont le but est d'avoir une vision globale sur le réseau. C'est l'entité Network_CLE qui se charge de collecter et maintenir ces données et d'en déduire l'état global du réseau concernant une métrique précise. Comme exemple de ces métriques, nous citons la charge globale du réseau. Dans ce cas précis, si la charge du nœud, évaluée par Local_CLE, est supérieure à la charge du réseau, évaluée par Network_CLE, alors le nœud doit ajuster sa charge à celle que le réseau peut supporter. Toute métrique locale qui doit être comparée à sa métrique homologue évaluée par Network_CLE est dite *locale à vue sur le réseau*. La charge du nœud est par la suite une métrique locale à vue sur le réseau.

V. Implémentation de XLEngine

L'architecture XLEngine est dans sa phase de conception. Les challenges rencontrés pour la suite de sa conception sont essentiellement :

- analyser les différents paramètres des couches à communiquer à *CLIC* et étudier les relations de dépendances entre ces paramètres ;
- étudier l'impact de tout changement d'un paramètre sur les autres couches et les autres paramètres ;
- identifier un mécanisme permettant aux couches d'avoir une vue claire sur les données et les algorithmes maintenus par *CLIC*. Et permettant pour chaque couche de savoir quels sont les données et/ou les algorithmes qui la concerne;
- un autre challenge très important est de développer un mécanisme d'échange de données locales aux noeuds. Ces données doivent refléter au mieux l'état d'un noeud. Il faut également développer un mécanisme de traitement de ces données pour en déduire une vue globale sur le réseau.

Une fois l'architecture est entièrement décrite, l'étape suivante est sa validation formelle suivie par son évaluation.

VI. Conclusion

La suite de nos travaux de recherches sera orientée sur la description détaillée de notre architecture *XLEngine*. Cette étape sera suivie, dans un premier temps, par une validation formelle, puis par une implémentation suivie par son évaluation et une comparaison de ses performances avec l'architecture modulaire en couches isolées.

References

[1] J.Silva, M. Sgroi, F.Bernardinis, S.Li, A.Sangiovanni-Vincentelli, J. Rabaey. "Wireless Protocols Design: Challenges and Opportunities". Proceedings of the 8th IEEE International Workshop on Hardware/Software Codesign, S.Diego, CA, USA, May 2000, May, 2000.
[2] S.Krishnamachari, M.Schaar, S.Choi, X.Xu. "Video Streaming over Wireless LANs: A Cross-layer Approach". The 13th International Packetvideo Workshop. Nantes, France.
[3] M.Mirhakkak, N. Schult, and D.Thomson. "Dynamic bandwidth management and adaptive applications for a variable bandwidth wireless environment". IEEE J. Select. Areas Commun., vol. 19, pp. 1984–1997, Oct. 2001.
[4] J. Shin, J. Kim, and C.-C. Jay Kuo. "Quality-of-service mapping mechanism for packet video in differentiated services network". IEEE Trans. Multimedia, vol. 3, pp. 219–231, June 2001.
[5] W. Kumwilaisak, Y.T. Hou, Q. Zhang, W. Zhu, C.-C. Jay Kuo, and Ya-Qin Zhang. "A Cross-layer quality-of- service mapping architecture for video delivery in wireless networks". IEEE Journal on Selected Areas in Communications, vol. 21, no. 10, pp. 1685-1698, December 2003.
[6] Setton, E., Yoo, T., Zhu, X., Goldsmith, A., Girod, B., 2005. "Cross-layer design of ad-hoc networks for real-time video streaming". IEEE Wireless Communications, 12(4):59-65.
[7] S.Lohier, "Approches inter-couches pour la qualité de service dans les réseaux locaux sans fil". Thèse de doctorat, université de Pierre et Marie Curie, Paris – France. Juin 2006.
[8] V. Srivastava and M. Motani. "Cross-layer design: a survey and the road ahead". IEEE Communications Magazine, vol.43, no.12, pp.112-119, December 2005.
[9] V. T. Raisinghani and S. Iyer. "Cross layer design optimizations in wireless protocol stacks". Computer Communications vol.27 (8), pp.720-725, May 2004.
[10] V. Kawadia and P. R. Kumar. "A cautionary perspective on cross layer design". IEEE Wireless Communications", 12(1):3–11, February 2005.
[11] Pedro José Marrón, Daniel Minder, Andreas Lachenmann, and Kurt Rothermel. "TinyCubus: An Adaptive Cross-Layer Framework for Sensor Networks". it - Information Technology, vol. 47(2), pp. 87-97, 2005.
[12] R. Winter, J. Schiller, N. Nikaein, and C. Bonnet. "CrossTalk: Cross-layer decision support based on global knowledge". IEEE Communications Magazine, vol. 44, pp. 2–8, january 2006.
[13] Vijay T. Raisinghani, Sridhar Iyer. "ECLAIR: An Efficient Cross Layer Architecture for Wireless Protocol Stacks". 5th World Wireless Congress, San Francisco, USA, May 25-28 2004.

[14] Projet MobileMan: http://cnd.iit.cnr.it/mobileMAN/
[15] M. van der Schaar, S. Shankar. “Cross-layer Wireless Multimedia Transmission: Challenges, Principles and New Paradigms”. IEEE Wireless Communications Magazine, 12(4):50-58. August 2005.
[16] R. Sasanka, J. Srinivasan, and W. Yuan. “The Illinois GRACE Project: Global Resource Adaptation through CoopEration”. In Proceedings of the Workshop on Self-Healing, Adaptive, and self-MANaged Systems (SHAMAN) (held in conjunction with the 16th Annual ACM International Conference on Supercomputing), pp. 144*155, June 2002.
[17] K. Karenos, A. Khan, X. Chen, M. Faloutsos, and S. Krishnamurthy. “Local versus global power adaptive broadcasting in ad hoc networks”. In Proceedings of IEEE WCNC, New Orleans, MI, USA, March 2005.
[18] R. Knopp et al. “Overview of the WIDENS Architecture, A Wireless Ad Hoc Network for Public Safety”. 1st IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON), Santa Clara, USA, October 2004.